Power-Constrained Block-Test List Scheduling

Valentin Mureşan, Xiaojun Wang
Dublin City University, Ireland
muresanv@eeng.dcu.ie

Valentina Mureşan, Mircea Vlăduţiu
"Politehnica" University of Timişoara, România
vmuresan@cs.utt.ro

Abstract

A list scheduling approach is proposed in this paper to overcome the problem of unequal-length block-test scheduling under power dissipation constraints. An extended tree growing technique is also used in combination with the list scheduling algorithm in order to improve the test concurrency having assigned power dissipation limits. Moreover, the algorithm features a power dissipation balancing provision. Test scheduling examples are discussed highlighting further research steps towards an efficient system-level test scheduling algorithm.

1 INTRODUCTION

VLSI devices running in test mode can consume 100 - 200% higher power than when running in normal mode [1]. The heat dissipated during test application is lately one of the major considerations in *test scheduling*. Test scheduling is strongly related to test concurrency. *Test concurrency* is a design property which strongly impacts *testability* and *power dissipation*. To satisfy high fault coverage goals with *reduced test application time* under certain *power dissipation constraints*, the testing of all components on the system should be performed in parallel to the greatest extent possible.

The current paper brings under focus the high-level power-constrained block-test scheduling problem which lacks of practical solutions. An efficient scheme for overlaying the block-tests, called *extended tree growing technique* is employed together with the *list scheduling*, to search for power-constrained block-test schedule profiles in a polynomial time. The algorithm fully exploits test parallelism under power dissipation constraints. This is achieved by overlaying the block-test intervals of compatible subcircuits to test as many of them as possible concurrently so that the maximum accumulated power dissipation does not go over the given limit. A *constant additive model* is employed for the power dissipation estimation throughout the algorithm.

2 TEST SCHEDULING PROBLEM

The components which are required to perform a test (test control logic, test pattern generators, signature analyzers, test buses) are known as *test resources* and they may be shared among the blocks under test (BUT). Each activity or the ensemble of activities requiring a clock period during the *test mode* and occurring in the same clock period, can be considered as a *test step*. A *block test* is the sequence of test steps that correspond to a specific part of hardware (block). The testing of a VLSI system can be viewed as the execution of a collection of block tests. The steps in a step sequence belonging to the same block test can be pipelined and steps from different block tests can be executed concurrently, obviously if there are no resource conflicts between the steps. Two major types of test parallelism approaches have been identified in the literature thus far:

- *block-test scheduling*, which deals with *tests for blocks of logic* [2]. These potentially consist of many test vectors and are regarded as indivisible entities for test scheduling. It deals with more abstract descriptions (from register transfer (RT) to system levels);
- *test pipelining*, which deals with *test steps* that need to be applied and resources to be utilized in a specific temporal order [3]. It is applied at lower levels of abstraction, where the structure of the datapath is known in detail (logic or RT level);

Block tests and *test steps* have their *resource sets* used to build up their test plans. Depending on the test design methodology selected, once a *resource set* is compiled for each test t_i, then it is possible to determine whether they could run in parallel without any resource conflict. A pair of tests that cannot be run concurrently is said to be *incompatible*. Each application of time compatible tests is called a *test session*, and the time required for a test session is often referred to as *test length*. From this point of view, circuits fall, in general, into two classes:

- circuits in which all tests are equal in length;
- circuits in which the tests are unequal in length.

0-7695-0668-2/00 $10.00 © 2000 IEEE

3 POWER-CONSTRAINED BLOCK-TEST SCHEDULING

3.1 PROBLEM FORMULATION

In practical circuits (e.g., MCMs) only a few blocks or modules are activated at a certain moment, under normal system operation, while other blocks are in the power-down mode to minimize the power dissipation. Under testing environment, however, in order to test the system in the shortest possible time, it is desirable to concurrently activate as many blocks as possible provided that the power dissipation limit of the system is not exceeded.

The instantaneous power, $p(t)$, is the power dissipation at any time instant t: $p(t) = i(t) \times v(t)$, where $i(t)$ and $v(t)$ are the instantaneous current and voltage in the circuit. If $p(t_i)$ is the power dissipation during test t_i and $p(t_j)$ is the power dissipation during test t_j, then the power dissipation of a test session consisting of just these two tests is the sum of the instantaneous powers of test t_i and t_j. Usually this instantaneous power is constrained to not exceed the maximum power dissipation limit, PD_{max}, if they were meant to be executed in the same test session. However, in reality the instantaneous power for each test vector is hard to obtain at high-levels since it depends, e.g., in a CMOS circuit on the number of zero-to-one and one-to-zero transitions, which in turn could be dependent on the order of execution of test vectors. Consequently, different test schedules will result in different instantaneous power dissipation profiles for the same test.

A *constant additive model* is employed here. For high-level approaches the power dissipation $P(t_i)$ of a test t_i could be estimated in three ways. Firstly, $P(t_i)$ can be defined as the *average power dissipation* over all test vectors in t_i. This definition might be overly optimistic in the analysis of power dissipation when many test vectors are applied simultaneously, since the average value cannot reflect the instantaneous power dissipation of each test vector. Hence, it might lead to an undesirable test schedule which exceeds the power dissipation allowance of the device at some time instants. Secondly, $P(t_i)$ can be defined as the *maximum power dissipation* over all test vectors in t_i. This is the upper bound power dissipation in t_i and its definition is pessimistic in this case since it disallows two tests t_i and t_j, whose peak powers occur at different time instants, from being scheduled in the same test session. However, the test schedule obtained with this definition guarantees the maximum power dissipation allowance of the device to be observed at all time instants. Thirdly, an *RMS power dissipation* can be used when the instantaneous power dissipation is prone to power spikes and a more accurate estimation is sought. The engineer has the freedom to choose any of the above ways to be run with the algorithm proposed here.

3.2 PREVIOUS WORK VS CURRENT WORK

Power dissipation during test scheduling was seldom under research so far. [4] proposes for the first time only a theoretical analysis of this problem at IC level. It is, basically, a compatible test clustering, where the compatibility among tests is given by test resource and power dissipation conflicts at the same time. Unfortunately, from an implementation point of view the identification of all cliques in the graph of compatible block-tests belongs to the class of NP-hard problems. Instead, a greedy approach is proposed in this paper. It has a polynomial complexity, which is very important for the success of a fast system-level test scheduling solution. A *list scheduling* together with a *tree growing technique* are employed here to generate the block-test schedule profile at the node level, within a test hierarchy.

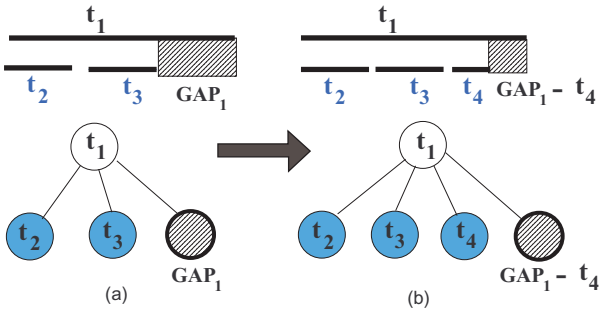


Figure 1. Merging Step Example

The proposed algorithm is an *unequal-length block-test scheduling* one [2] because it deals with tests for blocks of logic, which do not have equal test lengths. It is meant to be part of a system-level block-test approach to be applied on a modular view of a test hierarchy. The modular elements of this hierarchy could be given at any of the high-level synthesis (HLS) domains, between the system and RT levels: subsystems, backplanes, boards, MCM's, IC's (dies), macro blocks and RTL transfer blocks. The lowest level block the test hierarchy accepts is the RTL one, but at this level it is assumed that a test-step level scheduling has already been taken into consideration and applied. Generally speaking any node in hierarchy (apart from leaves) has different subnodes as children. Every test node t_i is characterized by a few parameters, which it was previously assigned with, after the test scheduling optimization has been applied on it. These features are the following: test application time T_i, power dissipation P_i, and test resource set RES_SET_i. This approach assumes a bottom-up traversing of the hierarchical test model within a *divide et impera* optimization style. Thus, at a certain moment the subnodes of a certain node are considered for optimization in order to get an optimal or near optimal sequencing or overlaying of them complying with the power dissipation constraints.

0-7695-0668-2/00 $10.00 © 2000 IEEE

4 TREE GROWING TECHNIQUE

Since the block-test set in a complex VLSI circuit is huge, it is possible to schedule some short tests to begin, if they are resource compatible, when subcircuits with shorter testing time have finished testing, while other subcircuits with longer testing time have not. The *tree growing technique* given in [5] is very productive from this point of view. That is because it is used to exploit the potential of test parallelism by merging and constructing the *concurrent testable sets* (CTS). This was achieved by means of a *binary tree structure* (not necessarily complete), called *compatibility tree*, which was based on the compatibility relations among the tests.

Nevertheless, a big drawback in [5] is that the compatibility tree is a binary one. This limits the number of children test nodes that could be overlapped to the parent test node to only two. In reality the number of children test nodes can be much bigger, as in the examples depicted in figures 1 and 2. Therefore, an *extended compatibility tree* (ECT), given by means of a *generalized tree*, is proposed here to overcome this problem. Figure 2 gives the *test schedule chart* and the ECT for the *power-test scheduling chart* example depicted in figure 3(a) and presented in section 6. The sequence of nodes contained in the same tree path represents an expansion of the CTS. Given a partial schedule chart of a CTS, a test t can be merged in this CTS if and only if there is at least one tree path P in the corresponding compatibility tree of CTS, such that every test contained in the nodes of P is compatible to t. The compatibility relation here has three components. Firstly, tests have to be compatible from a conflicting resources point of view. Secondly, the test length of the nodes in a tree path have to be monotonously growing from leaf to root. Thirdly, the power dissipation accumulated on the above tree path is less than or equal to PD_{max}.

A *merging step* example is given in figure 1. Partial test schedule charts are given at the top, while partially grown compatibility trees are given at the bottom. Suppose tests t_2, t_3 and t_4 are compatible to t_1, while they are not compatible to each other. Suppose T_1, T_2, T_3 and T_4 are, respectively, the test lengths of tests t_1, t_2, t_3 and t_4, and say $T_2 + T_3 < T_1$. Suppose now, a new test t_4 has to be scheduled in parallel to the partial test schedule depicted in figure 1(a). As can be seen, there is a gap GAP_1 given by the following test length difference: $GAP_1 = T_1 - (T_2 + T_3)$. Thus a merging step can be achieved, if $T_4 \leq GAP_1$, by inserting t_4 in the partial test schedule and its associated ECT in figure 1(b).

The process of constructing CTSs can be implemented by expanding (growing) the ECT from the roots to their leaf nodes. The root nodes are considered test sessions, while the expanded tree paths are considered their test subsessions. When a new test has to be merged with the CTS,

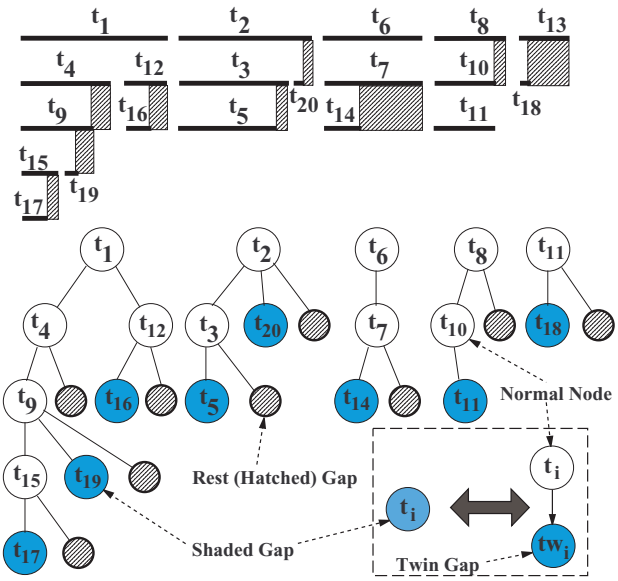


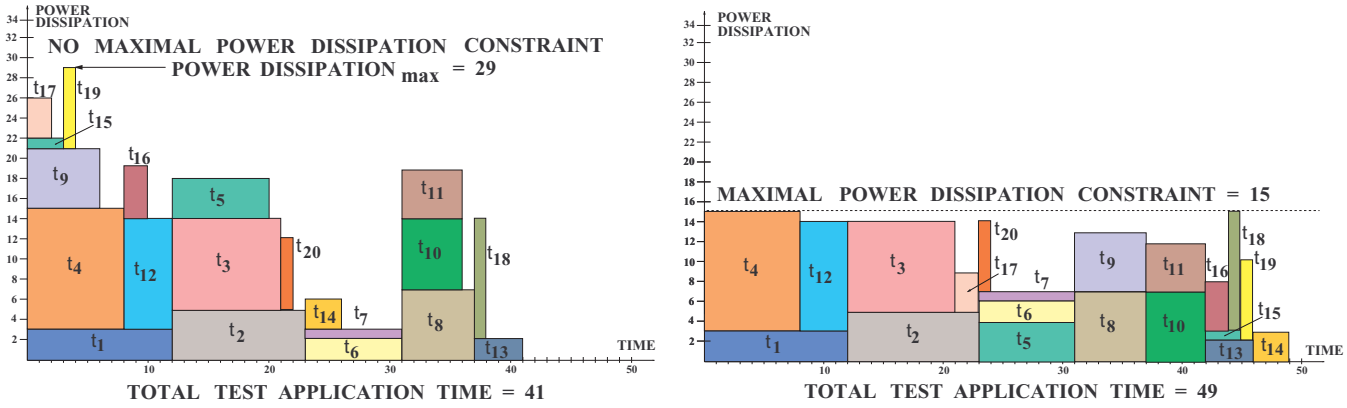
Figure 2. Tree Growing Example

the algorithm should avail of all possible paths in the ECT. In order to keep track of the available tree paths and to avoid the complexity of the generalized tree travel problem, a list of potentially *expandable tree paths* (ETP) is kept. This list is kept by means of special nodes that are inserted as leaf nodes within each ETP of ECT. These leaf nodes are called *gaps* and are depicted as hatched or shaded nodes in figures 1 and 2. There are two types of gaps. The first set of gaps (hatched) are those "rest gaps" left behind each merging step, like it was the case of GAP_1 and $GAP_1 - t_4$ in the above example. They are similar to the uncomplete branches of the binary tree from [5]. The second set of gaps (shaded), are actually bogus gaps generated as the superposition of the leaf nodes and their twins as in the bottom-right equivalence given in figure 2. The twin gaps are generated in order to keep track of "non-saturated" tree paths, which are also potential ETP's. By "non-saturated" tree path is meant any ETP who's accumulated power dissipation is still under the given power dissipation limit. The root nodes (test sessions) are considered by default "shaded" gaps before any test subsession is generated below them. Finally, the test scheduling chart (figure 2) can easily be expanded to a power-test scheduling chart (figures 3) to asses the power dissipation distribution over the test schedule.

5 POWER-CONSTRAINED BLOCK-TEST LIST SCHEDULING

The biggest achievement of the tree growing technique is that HLS algorithms proved to be efficient can be easily applied on the power-constrained test scheduling (PTS)

0-7695-0668-2/00 $10.00 © 2000 IEEE



(a) Without Power Dissipation Constraints

(b) With Maximal Power Dissipation Constraint = 15

Figure 3. Power-Test Scheduling Charts Of The List Scheduling Approach

problem modeled as an extended tree growing process. This is due to the high degree of similarities between the HLS tasks, e.g. HLS scheduling, and power-constrained block-test scheduling modeled as a growing tree problem. Classical HLS approaches like the left-edge algorithm for HLS register allocation have already been successfully applied on power-constrained block-test scheduling in [7]. In this paper the efficiency of the HLS list scheduling algorithm (HLS-LS) in test scheduling is presented and is named *power-constrained block-test list scheduling* (PTS-LS). Moreover, the same tree growing technique is currently experienced in combination with other classical HLS scheduling algorithms like the force-directed based ones : force-directed scheduling, improved force-directed scheduling, and force-directed list scheduling.

In the HLS-LS approach [6], operations were sorted in topological order by using control and data dependencies. The set of operations that could be placed in a control step (c-step) were then evaluated. These operations were called *ready* operations. If the number of ready operations of a single type exceeded the number of hardware modules available to perform them, then one or more operations had to be deferred. In previous list scheduling algorithms, the selection of the deferred operations was determined by a local priority function such as *mobility* or *urgency*.

Since every block-test t_i in our approach has a test length T_i and a power dissipation P_i, a local priority function called *test mobility* TM_i can also be defined for t_i as the inverse of the product between its test length T_i and its power dissipation P_i: $TM_i = \frac{1}{T_i * P_i}$. Intuitively, the probability of scheduling a block-test into a test subsession is higher the higher block-test's mobility TM_i is. The mobility of a block-test t_i is inverse proportional with its dimensions. The dimension of a block-test t_i is given by the rectangle area having its test length T_i and its power dissipation P_i as sides. That is, bigger is the area of the rectangles associated with the block-tests, smaller mobility they get. Though, in the tree growing approach the block-tests in an ETP are monotonously growing in terms of test length. A block-test cannot be scheduled in an ETP, where the leaf's test length is shorter than the test length of the block-test to be scheduled. This is due to the fact that the block-tests have to be scheduled in an ETP in the order of their test lengths. Therefore the mobility in the PTS-LS approach is split into two, its test length component and its power dissipation component. Thus, in PTS-LS, the block-tests are sorted in topological order by using the test length as primary key to order in descending order, and the power dissipation as secondary key, to order the block-tests having the same test length in descending order as well.

A clear parallel between the HLS scheduling problem and the PTS problem is given by the similarities between the c-steps from HLS and the test sessions (test subsessions) in PTS, between operations (HLS) and block-tests (PTS), and between hardware resource constraints (HLS) and power dissipation constraints (PTS). Therefore, there is an obvious coincidence between the process of assigning operations to c-steps (HLS scheduling) and the process of assigning block-tests to test (sub)sessions (PTS). In the current PTS-LS algorithm the block-tests are initially ordered (as described above) before being scheduled. The sorted block-tests are then iteratively scheduled into the available test (sub)sessions (ETPs). When the power dissipation is exceeded the block-tests to be currently scheduled are deferred for the other test (sub)sessions (ETP) left for further expansion. In terms of test scheduling, the list scheduling approach given here (PTS-LS) resembles the first algorithm of the PTS approach given in [7]. An iterative tree growing technique was also proposed there to minimize the total test application time by searching for every block-test an ETP to

0-7695-0668-2/00 $10.00 © 2000 IEEE

be assigned to. The assignment was achieved if and only if the accumulated power dissipation in the just expanded tree path was not exceeding the given power dissipation constraint.

The PSEUDOCODE of the PTS-LS ALGORITHM:
-sort all the block-tests by their mobility in two steps (test length, power consumption);
-initialize the $GrowingTree$, the $BlockTestList$ and the $GapsList$;
-initialize the time constraint to the initial number of test subsessions;
-$GapsList$ is ordered by its gaps' power consumption;
-while (there are unscheduled block-tests) do:
/*$BlockTestList$ is not empty*/ {

- **if ($GapsList$ is empty) then** {
  - $CurTest$ **= head of** $BlockTestList$**;**
  - **insert** $CurTest$ **as the tail of** $GrowingTree$ **roots (new test section);**
  - **make** $CurTest$ **"used";**
  - **remove** $CurTest$ **from** $BlockTestList$**;**
  - **generate a** $TwinGap$ **gap as the twin of** $CurTest$**;**
  - **insert** $TwinGap$ **into** $GapsList$ **};**
- **else** {
  - $CurGap$ **= the head of** $GapsList$**;**
  - $CurTest$ **= the head of** $Comp.List_{CurGap}$**;**
  - **while (**$T_{CurTest} > T_{CurGap}$ **OR** $PD_{CurGap} + PD_{CurTest} > PD_{max}$ **OR** $CurTest$ **"used") do:** {
    - $CurTest = CurTest -\rightarrow next$ **(next in the** $Comp.List_{CurGap}$**);**} **/*while*/**
  - **if (**$T_{CurTest} \leq T_{CurGap}$ **AND** $PD_{CurGap} + PD_{CurTest} \leq PD_{max}$ **AND** $CurTest$ **NOT "used") then**
    - **SCHEDULE(**$CurTest$**,**$CurGap$**,** $GrowingTree$**,**$GapsList$**,**$BlockTestList$**);**
  - **else remove** $CurGap$ **from** $GapsList$**;**

-}/*while*/

The complexity of this approach is $O(n^2)$. The data structures used in it are: the $Growing\ Tree$ to model the ECT, $GapsList$ to model the ordered list of potentially expandable gaps (shaded and hatched gaps), $BlockTestList$ to keep the ordered but not yet merged block-tests. $CurTest$ is the block-test to be merged at a certain iteration. $CurGap$ is the gap under focus at a certain iteration to see whether it is expandable (compatible) with the $CurTest$. In the pseudocode "used" means that the block-test has already been merged in the ECT. $TwinGap$ is the newly generated shaded gap at every iteration and it will not be inserted in the $GapsList$ anymore after its generation, if its resulting compatibility list is null, i.e. it will not be an ETP. $RestGap$ is meant to keep the hatched gap generated at every iteration if it is not null, i.e. $CurTest$ covers completely $CurGap$.

The algorithm is iterative. Every iteration looks for the block-test with biggest test length and lowest mobility to be scheduled in the test subsession (ETP) having the lowest accumulated power dissipation. Thus, $CurGap$ (current gap) is assigned all the time with the first gap from the $GapsList$, which is the one with the lowest accumulated power dissipation. Next, $CurGap's$ compatibility list is crossed from the block-test with the lowest mobility to the one with the highest mobility. The first block-test ($CurTest$) which is found assignable (compatible from all points of view) to the $CurGap$ is scheduled in the current gap. Another pair of gaps (twin and rest) are generated and they have to be inserted in the $GapsList$ so that the list is still ordered by gaps' accumulated power dissipation. If no block-test is found for further scheduling in the $CurGap's$ compatibility list, then $CurGap$ is removed from the $GapsList$ and the algorithm continues with the new head of the list. Otherwise, the new head of $GapsList$ is considered for expansion. When all the gaps from the $GapsList$ are removed a new test session is generated exactly like at the beginning of the algorithm during the initialization of the $GapsList$. Namely, the $GapsList$ is set to the first (biggest test length) block-test left unused in the sorted $BlockTestList$. The *SCHEDULE* procedure carries out the insertion of block-test to be merged into the $GrowingTree$ and subsequently its removal from the $BlockTestList$. For space reasons it is not detailed in this paper. As can be seen in figure 1, the merging step implies the generation of shaded and hatched gaps. The SCHEDULE procedure in the above pseudocode is similar to the SCHEDULE procedures of the other tree growing based approaches [7], but the current one features a power dissipation balancing provision. It keeps the gaps in the $GapsList$ in a increased order of their accumulated power dissipation. This is implemented in order to select at every iteration for further expansion the test subsessions with lower accumulated power dissipation. Thus, there is a higher probability of decreasing the power dissipation difference between the test subsessions (ETPs) of the growing tree. Consequently, the power dissipation would be more balanced.

6 EXPERIMENTAL RESULTS

The following example should provide a deeper insight into how this algorithm works and its results. Suppose the following block-tests are to be scheduled and their parameters specified in the order: power dissipation, test length and their compatibility list. For simplicity reasons, the block tests are already ordered by test length and test mobility as described in section 5.

t_1 (3, 12, $\{t_4, t_5, t_8, t_9, t_{10}, t_{12}, t_{15}, t_{16}, t_{17}, t_{19}, t_{20}\}$)
t_2 (5, 11, $\{t_3, t_4, t_5, t_9, t_{12}, t_{13}, t_{14}, t_{17}, t_{19}, t_{20}\}$)
t_3 (9, 9, $\{t_2, t_5, t_7, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{17}, t_{18}\}$)
t_4 (12, 8, $\{t_1, t_2, t_7, t_9, t_{11}, t_{14}, t_{15}, t_{17}, t_{19}\}$)
t_5 (4, 8, $\{t_1, t_2, t_3, t_6, t_7, t_8, t_{12}, t_{15}, t_{17}, t_{18}, t_{20}\}$)

0-7695-0668-2/00 $10.00 © 2000 IEEE

t_6 (2, 8, $\{t_5, t_7, t_9, t_{11}, t_{14}, t_{17}, t_{20}\}$)
t_7 (1, 8, $\{t_3, t_4, t_5, t_6, t_9, t_{12}, t_{14}, t_{15}, t_{16}, t_{18}, t_{19}, t_{20}\}$)
t_8 (7, 6, $\{t_1, t_5, t_9, t_{10}, t_{11}, t_{14}, t_{16}, t_{17}, t_{19}, t_{20}\}$)
t_9 (6, 6, $\{t_1, t_2, t_4, t_6, t_7, t_8, t_{11}, t_{12}, t_{15}, t_{17}, t_{19}\}$)
t_{10}(7, 5, $\{t_1, t_3, t_8, t_{11}, t_{15}, t_{16}, t_{17}, t_{18}\}$)
t_{11}(5, 5, $\{t_3, t_4, t_6, t_8, t_9, t_{10}, t_{14}, t_{16}, t_{18}, t_{20}\}$)
t_{12}(11, 4, $\{t_1, t_2, t_3, t_5, t_7, t_9, t_{13}, t_{14}, t_{16}, t_{19}\}$)
t_{13}(2, 4, $\{t_2, t_3, t_{12}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}\}$)
t_{14}(3, 3, $\{t_2, t_3, t_4, t_6, t_7, t_8, t_{11}, t_{12}, t_{16}, t_{18}, t_{20}\}$)
t_{15}(1, 3, $\{t_1, t_4, t_5, t_7, t_9, t_{10}, t_{13}, t_{16}, t_{17}, t_{18}\}$)
t_{16}(5, 2, $\{t_1, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{17}, t_{19}, t_{20}\}$)
t_{17}(4, 2, $\{t_1, t_2, t_3, t_4, t_5, t_6, t_8, t_9, t_{10}, t_{13}, t_{15}, t_{16}, t_{18}, t_{19}, t_{20}\}$)
t_{18}(12, 1, $\{t_3, t_5, t_7, t_{10}, t_{11}, t_{13}, t_{14}, t_{15}, t_{17}, t_{19}, t_{20}\}$)
t_{19}(8, 1, $\{t_1, t_2, t_4, t_7, t_8, t_9, t_{12}, t_{13}, t_{16}, t_{17}, t_{18}, t_{20}\}$)
t_{20}(7, 1, $\{t_1, t_2, t_5, t_6, t_7, t_8, t_{11}, t_{14}, t_{16}, t_{17}, t_{18}, t_{19}\}$)

In figure 3 the results of the PTS-LS algorithm are given both with (figure 3(a)) and without (figure 3(b)) power dissipation constraints ($PD_{max} = 15$). Table 1 gives the results of the PTS-LS algorithm for a randomly generated 50 block-tests set. Their degree of resource compatibility has been increased within a range from low to high: low (L) 10%, average-low (AL) 30%, average (AV) 50%, average-high (AH) 70% and high (H) 90%. The following abbreviations have been used in the table: test length (TL), maximal accumulated power dissipation (MPD), average power dissipation (APD) and the total power dissipation dispersion (PDD). TL represents the total test application time of the test scheduling solutions. MPD is the maximal power dissipation over the power dissipations accumulated in all ETPs of the final ECT (growing tree). APD is considered the ideal MPD when all the ETP's would exhibit the same accumulated power dissipation, that is the power dissipation would be fully balanced over the power-test scheduling chart. It is calculated as the ratio between the power-test area, taken up by the chart (see figure 3) of the ECT, and the TL. The rectangle given by APD and TL would be the ideal power-test scheduling chart and, therefore, the ideal test schedule profile. PDD is direct proportional to the accumulated power dissipation dispersion over the power-test scheduling chart, which is considered here to be given by the power-test area left unused inside the power-test rectangle given by MPD and TL. PDD is calculated as the difference between MPD and APD. The smaller the PDD values of the PTS-LS solutions are the better representativeness the MPD value gets.

It can be seen in figure 3(b) and table 1 that a tighter power dissipation constraint forces the test scheduling to a more balanced power dissipation throughout the test application time. At the same time obvious power dissipation spikes could be seen in figure 3(a) due to the lack of power dissipation constraints. That means the power dissipation is less balanced when it is loosely constrained. On the other hand when there are tighter power dissipation constraints (see table 1), the total test application time increases. Thus, the PTS-LS problem turned out to be a trade-off problem to be solved with more complex algorithms.

	$PD_{max}\ LIMIT = 200$				$PD_{max}\ LIMIT = 50$			
	TL	MPD	APD	PDD	TL	MPD	APD	PDD
L	401	33	15.9	17.1	401	33	15.9	17.1
AL	301	41	21.3	19.7	301	41	21.3	19.7
AV	204	79	31.4	47.6	224	50	28.6	21.4
AH	154	94	41.6	52.4	167	50	38.4	11.6
H	99	197	62.3	134.7	151	50	42.5	7.5

Table 1. Power-Test List Scheduling Results

7 CONCLUSIONS

This novel greedy unequal-length block-test scheduling approach is based on the classical list scheduling algorithm applied to an extended tree growing technique and its polynomial complexity is beneficial to the system-level test scheduling problem. Thus, this fast algorithm is very suitable to be part of a HLS methodology meant to obtain *rapid system prototyping* solutions. Even though it does not guarantee optimal block-test scheduling solutions, its final result can be used as a starting point by near-optimal block-test scheduling approaches (i.e., simulated annealing, genetic algorithms, tabu search) to get an improved solution.

References

[1] Y. ZORIAN: **A Distributed BIST Control Scheme for Complex VLSI Devices** - *Proceedings of The 11th IEEE VLSI Test Symposium*, pp. 4-9, Apr, 1993.

[2] G.L. CRAIG, C.R. KIME, K.K. SALUJA: **Test Scheduling and Control for VLSI Built-In Self-Test** - *IEEE Transactions on Computer*, Vol. 37, No. 9, pp. 1099–1109, Sep, 1988.

[3] M. ABADIR, M. BREUER: **Test Schedules for VLSI Circuits Having Built-in Test Hardware** - *IEEE Transactions on Computer*, Vol. C-35, No. 4, pp. 361–368, Apr, 1986.

[4] R.M. CHOU, K.K. SALUJA, V.D. AGRAWAL: **Scheduling Tests for VLSI Systems Under Power Constraints** - *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 5, No. 2, pp. 175–185, Jun, 1997.

[5] W.B. JONE, C. PAPACHRISTOU, M. PEREIRA: **A Scheme for Overlaying Concurrent Testing of VLSI Circuits** - *Proceedings of the 26th Desing Automation Conference*, pp. 531-536, 1989.

[6] S. DAVIDSON et al: **Some Experiments in Local Microcode Compaction for Horizontal Machines** - *IEEE Transactions on Computers*, Vol. C-30, No. 7, pp. 460–477, Jul, 1981.

[7] V. MURESAN, X. WANG, V. MURESAN, M. VLADUTIU: **The Left Edge Algorithm and the Tree Growing Technique in Power-Constrainted Block-Test Scheduling** - *Proceedings of the 18th IEEE VLSI TEST SYMPOSIUM, 2000*, accepted paper, Montreal, May, 2000.

0-7695-0668-2/00 $10.00 © 2000 IEEE